Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	312	(712/209).CCLS.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	OFF	2005/09/06 08:03
S1	299	(712/209).CCLS.	US-PGPUB; USPAT; EPO; JPO; IBM_TDB	OR	OFF	2005/02/22 07:50

Web

Results **1** - **10** of about **32,500** for **ARM THUMB instruction format + 1999**. (**0.20** seconds)

ARM ASSEMBLER PROGRAMMING; tutorial, resources, and examples
Please note, however, that the '**Thumb**' **instruction** set is not (yet) described.
... **Instruction** Quick Finder - lists all **ARM instructions** I'm aware of ...
www.heyrick.co.uk/assembler/ - 26k - Cached - Similar pages

ACM: Digital Library: Communications of the ACM
In **ARM** state, 32-bit **instructions** are executed, and in **Thumb** state, ... The **ARM** ISA (**instruction** set architecture) supports a three-address **format**, ...
portal.acm.org/ft_gateway.cfm?id=859697&type=html - Similar pages

> Dynamic coalescing for 16-bit **instructions**
> 9 Jacobson, Q. and Smith, JE **1999**. **Instruction** pre-processing in trace processors. ... Profile guided selection of **arm** and **thumb instructions**. ...
> portal.acm.org/citation.cfm?id=1053271.1053273 - Similar pages

[PDF] Heads and Tails: A Variable-Length **Instruction Format** Supporting ...
File Format: PDF/Adobe Acrobat - View as HTML
Existing variable-length **instruction formats** provide higher code ... The **ARM Thumb** [18] and MIPS16 [13] **instruction** sets provide ...
www.cag.csail.mit.edu/scale/papers/hat-cases2001.pdf - Similar pages

EETimes.com - Free 32-bit processor core hits the Net
Lampret said the initial goal had been to publish the core in **1999** and to port ... or 16-to-32-bit **instruction format** translation (used in **ARM's Thumb** and ...
www.eetimes.com/story/OEG20000228S0007 - 65k - Cached - Similar pages

[PDF] Dynamic Coalescing for 16-Bit **Instructions**
File Format: PDF/Adobe Acrobat - View as HTML
and 16-bit **Thumb instruction** sets. The data in Figure 1 compares the **ARM** and ... for the lack of three address **instruction format** in **Thumb**. We introduce the ...
www.cs.arizona.edu/people/gupta/ research/Publications/Comp/16tecs.pdf - Similar pages

[PDF] **Thumb Instruction** Set Quick Reference Card
File Format: PDF/Adobe Acrobat - View as HTML
Change to **ARM**. Encoded as two **Thumb instructions**. ... FPSCR **format**. Rounding. (Stride – 1)*3. Vector length – 1 ... Oct **1999**. CKS. Fourth Release ...
www.**arm**.com/pdfs/QRC0001H_rvct_v2.1_**thumb**.pdf - Similar pages

Citations: MIPS16: High-density MIPS for the Embedded Market ...
The **ARM Thumb** [25] and MIPS16 [20] **instruction** sets provide alternate 16 bit ... from the short **instruction format** to the wider **instruction format** in the ...
citeseer.ist.psu.edu/context/425491/0 - 35k - Cached - Similar pages

> Citations: An Introduction to **Thumb** - Machines (ResearchIndex)
> **ARM** and MIPS are examples of such dual mode **instruction** sets. ... The **Thumb instruction** set is essentially a new **instruction** set which consists of a subset ...
> citeseer.ist.psu.edu/context/425487/0 - 16k - Cached - Similar pages

[PDF] Thomas J
File Format: PDF/Adobe Acrobat - View as HTML
instruction sequences that can be replaced profitably with better. sequences. ... commercial C compiler for **ARM** Ltd.'s **ARM/Thumb** family of ...

Thumb® Instruction Set
Quick Reference Card

Key to Tables				
<loreglist>	A comma-separated list of Lo registers, enclosed in braces, { and }.		<loreglist+LR>	A comma-separated list of Lo registers, plus the LR, enclosed in braces, { and }.
			<loreglist+PC>	A comma-separated list of Lo registers, plus the PC, enclosed in braces, { and }.

All Thumb registers are Lo (R0-R7) except where specified. Hi registers are R8-R15.

Operation		§	Assembler	Updates	Action	Notes
Move	Immediate		MOV Rd, #<immed>	N Z	Rd := immed	Immediate range 0-255.
	Lo to Lo		MOV Rd, Rm	N Z * *	Rd := Rm	* Clears C and V flags.
	Hi to Lo, Lo to Hi, Hi to Hi		MOV Rd, Rm		Rd := Rm	Not Lo to Lo. Flags not affected.
	Copy Any to Any	6	CPY Rd, Rm		Rd := Rm	Any register to any register. Flags not affected.
Arithmetic	Add		ADD Rd, Rn, #<immed>	N Z C V	Rd := Rn + immed	Immediate range 0-7.
	Lo and Lo		ADD Rd, Rn, Rm	N Z C V	Rd := Rn + Rm	
	Hi to Lo, Lo to Hi, Hi to Hi		ADD Rd, Rm		Rd := Rd + Rm	Not Lo to Lo. Flags not affected.
	immediate		ADD Rd, #<immed>	N Z C V	Rd := Rd + immed	Immediate range 0-255.
	with carry		ADC Rd, Rm	N Z C V	Rd := Rd + Rm + C-bit	
	value to SP		ADD SP, #<immed>		R13 := R13 + immed	Immediate range 0-508 (word-aligned). Flags not affected.
	form address from SP		ADD Rd, SP, #<immed>		Rd := R13 + immed	Immediate range 0-1020 (word-aligned). Flags not affected.
	form address from PC		ADD Rd, PC, #<immed>		Rd := (R15 AND 0xFFFFFFFC) + immed	Immediate range 0-1020 (word-aligned). Flags not affected.
	Subtract		SUB Rd, Rn, Rm	N Z C V	Rd := Rn – Rm	
	immediate 3		SUB Rd, Rn, #<immed>	N Z C V	Rd := Rn – immed	Immediate range 0-7.
	immediate 8		SUB Rd, #<immed>	N Z C V	Rd := Rd – immed	Immediate range 0-255.
	with carry		SBC Rd, Rm	N Z C V	Rd := Rd – Rm – NOT C-bit	
	value from SP		SUB SP, #<immed>		R13 := R13 – immed	Immediate range 0-508 (word-aligned). Flags not affected.
	Negate		NEG Rd, Rm	N Z C V	Rd := – Rm	
	Multiply		MUL Rd, Rm	N Z * *	Rd := Rm * Rd	* C and V flags unpredictable in §4T, unchanged in §5T and above
	Compare		CMP Rn, Rm	N Z C V	update CPSR flags on Rn – Rm	Can be Lo to Lo, Lo to Hi, Hi to Lo, or Hi to Hi.
	negative		CMN Rn, Rm	N Z C V	update CPSR flags on Rn + Rm	
	immediate		CMP Rn, #<immed>	N Z C V	update CPSR flags on Rn – immed	Immediate range 0-255.
	No operation		NOP		None	Flags not affected.
Logical	AND		AND Rd, Rm	N Z	Rd := Rd AND Rm	
	Exclusive OR		EOR Rd, Rm	N Z	Rd := Rd EOR Rm	
	OR		ORR Rd, Rm	N Z	Rd := Rd OR Rm	
	Bit clear		BIC Rd, Rm	N Z	Rd := Rd AND NOT Rm	
	Move NOT		MVN Rd, Rm	N Z	Rd := NOT Rm	
	Test bits		TST Rn, Rm	N Z	update CPSR flags on Rn AND Rm	
Shift/rotate	Logical shift left		LSL Rd, Rm, #<shift>	N Z C*	Rd := Rm << shift	Allowed shifts 0-31. * C flag unaffected if shift is 0.
			LSL Rd, Rs	N Z C*	Rd := Rd << Rs[7:0]	* C flag unaffected if Rs[7:0] is 0.
	Logical shift right		LSR Rd, Rm, #<shift>	N Z C	Rd := Rm >> shift	Allowed shifts 1-32.
			LSR Rd, Rs	N Z C	Rd := Rd >> Rs[7:0]	* C flag unaffected if Rs[7:0] is 0.
	Arithmetic shift right		ASR Rd, Rm, #<shift>	N Z C	Rd := Rm ASR shift	Allowed shifts 1-32.
			ASR Rd, Rs	N Z C*	Rd := Rd ASR Rs[7:0]	* C flag unaffected if Rs[7:0] is 0.
	Rotate right		ROR Rd, Rs	N Z C*	Rd := Rd ROR Rs[7:0]	* C flag unaffected if Rs[7:0] is 0.
Reverse	Bytes in word	6	REV Rd, Rm		Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24]	
	Bytes in both halfwords	6	REV16 Rd, Rm		Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]	
	Bytes in low halfword, sign extend	6	REVSH Rd, Rm		Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF	

Operation		§	Assembler	Action	Notes
Load	with immediate offset, word		`LDR Rd, [Rn, #<immed>]`	Rd := [Rn + immed]	Immediate range 0-124, multiple of 4.
	halfword		`LDRH Rd, [Rn, #<immed>]`	Rd := ZeroExtend([Rn + immed][15:0])	Clears bits 31:16. Immediate range 0-62, even.
	byte		`LDRB Rd, [Rn, #<immed>]`	Rd := ZeroExtend([Rn + immed][7:0])	Clears bits 31:8. Immediate range 0-31.
	with register offset, word		`LDR Rd, [Rn, Rm]`	Rd := [Rn + Rm]	
	halfword		`LDRH Rd, [Rn, Rm]`	Rd := ZeroExtend([Rn + Rm][15:0])	Clears bits 31:16
	signed halfword		`LDRSH Rd, [Rn, Rm]`	Rd := SignExtend([Rn + Rm][15:0])	Sets bits 31:16 to bit 15
	byte		`LDRB Rd, [Rn, Rm]`	Rd := ZeroExtend([Rn + Rm][7:0])	Clears bits 31:8
	signed byte		`LDRSB Rd, [Rn, Rm]`	Rd := SignExtend([Rn + Rm][7:0])	Sets bits 31:8 to bit 7
	PC-relative		`LDR Rd, [PC, #<immed>]`	Rd := [(R15 AND 0xFFFFFFFC) + immed]	Immediate range 0-1020, multiple of 4.
	SP-relative		`LDR Rd, [SP, #<immed>]`	Rd := [R13 + immed]	Immediate range 0-1020, multiple of 4.
	Multiple		`LDMIA Rn!, <reglist>`	Loads list of registers	Always updates base register.
Store	with immediate offset, word		`STR Rd, [Rn, #<immed>]`	[Rn + immed] := Rd	Immediate range 0-124, multiple of 4.
	halfword		`STRH Rd, [Rn, #<immed>]`	[Rn + immed][15:0] := Rd[15:0]	Ignores Rd[31:16]. Immediate range 0-62, even.
	byte		`STRB Rd, [Rn, #<immed>]`	[Rn + immed][7:0] := Rd[7:0]	Ignores Rd[31:8]. Immediate range 0-31.
	with register offset, word		`STR Rd, [Rn, Rm]`	[Rn + Rm] := Rd	
	halfword		`STRH Rd, [Rn, Rm]`	[Rn + Rm][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte		`STRB Rd, [Rn, Rm]`	[Rn + Rm][7:0] := Rd[7:0]	Ignores Rd[31:8]
	SP-relative, word		`STR Rd, [SP, #<immed>]`	[R13 + immed] := Rd	Immediate range 0-1020, multiple of 4.
	Multiple		`STMIA Rn!, <reglist>`	Stores list of registers	Always updates base register.
Push/ Pop	Push		`PUSH <loreglist>`	Push registers onto stack	Full descending stack.
	Push with link		`PUSH <loreglist+LR>`	Push LR and registers onto stack	
	Pop		`POP <loreglist>`	Pop registers from stack	
	Pop and return	4T	`POP <loreglist+PC>`	Pop registers, branch to address loaded to PC	
	Pop and return with exchange	5T	`POP <loreglist+PC>`	Pop, branch, and change to ARM state if address[0] = 0	
Branch	Conditional branch		`B{cond} label`	R15 := label	label must be within – 252 to + 258 bytes of current instruction. See Table **Condition Field** on reverse.
	Unconditional branch		`B label`	R15 := label	`label` must be within ±2Kb of current instruction.
	Long branch with link		`BL label`	R14 := address of next instruction, R15 := label	Encoded as two Thumb instructions. label must be within ±4Mb of current instruction.
	Branch and exchange		`BX Rm`	R15 := Rm AND 0xFFFFFFFE	Change to ARM state if Rm[0] = 0.
	Branch with link and exchange	5T	`BLX label`	R14 := address of next instruction, R15 := label Change to ARM	Encoded as two Thumb instructions. label must be within ±4Mb of current instruction.
	Branch with link and exchange	5T	`BLX Rm`	R14 := address of next instruction, R15 := Rm AND 0xFFFFFFFE	Change to ARM state if Rm[0] = 0
Extend	Signed extend halfword to word	6	`SXTH Rd, Rm`	Rd[31:0] := SignExtend(Rm[15:0])	
	Signed extend byte to word	6	`SXTB Rd, Rm`	Rd[31:0] := SignExtend(Rm[7:0])	
	Unsigned extend halfword to word	6	`UXTH Rd, Rm`	Rd[31:0] := ZeroExtend(Rm[15:0])	
	Unsigned extend byte to word	6	`UXTB Rd, Rm`	Rd[31:0] := ZeroExtend(Rm[7:0])	
Processor state change	Software interrupt		`SWI <immed_8>`	Software interrupt processor exception	8-bit immediate value encoded in instruction.
	Change processor state	6	`CPSID <iflags>`	Disable specified interrups	
		6	`CPSIE <iflags>`	Enable specified interrups	
	Set endianness	6	`SETEND <endianness>`	Sets endianness for loads and saves.	`<endianness>` can be `BE` (Big Endian) or `LE` (Little Endian).
	Breakpoint	5T	`BKPT <immed_8>`	Prefetch abort *or* enter debug state	8-bit immediate value encoded in instruction.

Vector Floating Point Instruction Set
Quick Reference Card

Key to Tables		
	{cond}	See Table **Condition Field**
	<S/D>	S (single precision) or D (double precision).
	<S/D/X>	As above, or X (unspecified precision).
	<VFPsysreg>	FPSCR, or FPSID.

Fd, Fn, Fm	Sd, Sn, Sm (single precision), or Dd, Dn, Dm (double precision).
{E}	E : raise exception on any NaN. Without E : raise exception only on signaling NaNs.
{Z}	Round towards zero. Overrides FPSCR rounding mode.
<VFPregs>	A comma separated list of *consecutive* VFP registers, enclosed in braces ({ and }).

Operation		Assembler	Exceptions	Action	Notes
Vector arithmetic	Multiply	FMUL<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := Fn * Fm	
	and negate	FNMUL<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := – (Fn * Fm)	
	and accumulate	FMAC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := Fd + (Fn * Fm)	
	negate and accumulate	FNMAC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := Fd – (Fn * Fm)	
	and subtract	FMSC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := – Fd + (Fn * Fm)	
	negate and subtract	FNMSC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := – Fd – (Fn * Fm)	
	Add	FADD<S/D>{cond} Fd, Fn, Fm	IO, OF, IX	Fd := Fn + Fm	
	Subtract	FSUB<S/D>{cond} Fd, Fn, Fm	IO, OF, IX	Fd := Fn – Fm	
	Divide	FDIV<S/D>{cond} Fd, Fn, Fm	IO, DZ, OF, UF, IX	Fd := Fn / Fm	
	Copy	FCPY<S/D>{cond} Fd, Fm		Fd := Fm	
	Absolute	FABS<S/D>{cond} Fd, Fm		Fd := abs(Fm)	
	Negative	FNEG<S/D>{cond} Fd, Fm		Fd := – Fm	
	Square root	FSQRT<S/D>{cond} Fd, Fm	IO, IX	Fd := sqrt(Fm)	
Scalar compare	Two values	FCMP{E}<S/D>{cond} Fd, Fm	IO	Set FPSCR flags on Fd – Fm	Use FMSTAT to transfer flags.
	Value with zero	FCMP{E}Z<S/D>{cond} Fd	IO	Set FPSCR flags on Fd – 0	Use FMSTAT to transfer flags.
Scalar convert	Single to double	FCVTDS{cond} Dd, Sm	IO	Dd := convertStoD(Sm)	
	Double to single	FCVTSD{cond} Sd, Dm	IO, OF, UF, IX	Sd := convertDtoS(Dm)	
	Unsigned integer to float	FUITO<S/D>{cond} Fd, Sm	IX	Fd := convertUItoF(Sm)	
	Signed integer to float	FSITO<S/D>{cond} Fd, Sm	IX	Fd := convertSItoF(Sm)	
	Float to unsigned integer	FTOUI{Z}<S/D>{cond} Sd, Fm	IO, IX	Sd := convertFtoUI(Fm)	
	Float to signed integer	FTOSI{Z}<S/D>{cond} Sd, Fm	IO, IX	Sd := convertFtoSI(Fm)	
Save VFP registers		FST<S/D>{cond} Fd, [Rn{, #<immed>}]		[address] := Fd. Immediate range 0-1020, multiple of 4.	
	Multiple, unindexed	FSTMIA<S/D/X>{cond} Rn, <VFPregs>		Saves list of VFP registers, starting at address in Rn.	
	increment after	FSTMIA<S/D/X>{cond} Rn!, <VFPregs>		synonym: FSTMEA (empty ascending)	
	decrement before	FSTMDB<S/D/X>{cond} Rn!, <VFPregs>		synonym: FSTMFD (full descending)	
Load VFP registers		FLD<S/D>{cond} Fd, [Rn{, #<immed>}]		Fd := [address]. Immediate range 0-1020, multiple of 4.	
	Multiple, unindexed	FLDMIA<S/D/X>{cond} Rn, <VFPregs>		Loads list of VFP registers, starting at address in Rn.	
	increment after	FLDMIA<S/D/X>{cond} Rn!, <VFPregs>		synonym: FLDMFD (full descending)	
	decrement before	FLDMDB<S/D/X>{cond} Rn!, <VFPregs>		synonym: FLDMEA (empty ascending)	
Transfer registers	ARM to single	FMSR{cond} Sn, Rd		Sn := Rd	
	Single to ARM	FMRS{cond} Rd, Sn		Rd := Sn	
	Two ARM to two singles	FMSRR{cond} {Sn,Sm}, Rd, Rn		Sn := Rd, Sm := Rn	Architecture VFPv2 only
	Two singles to two ARM	FMRRS{cond} Rd, Rn, {Sn,Sm}		Rd := Sn, Rn := Sm	Architecture VFPv2 only
	Two ARM to double	FMDRR{cond} Dn, Rd, Rn		Dn[31:0] := Rd, Dn[63:32] := Rn	Architecture VFPv2 only
	Double to two ARM	FMRRD{cond} Rd, Rn, Dn		Rd := Dn[31:0], Rn := Dn[63:32]	Architecture VFPv2 only
	ARM to lower half of double	FMDLR{cond} Dn, Rd		Dn[31:0] := Rd	Use with FMDHR.
	Lower half of double to ARM	FMRDL{cond} Rd, Dn		Rd := Dn[31:0]	Use with FMRDH.
	ARM to upper half of double	FMDHR{cond} Dn, Rd		Dn[63:32] := Rd	Use with FMDLR.
	Upper half of double to ARM	FMRDH{cond} Rd, Dn		Rd := Dn[63:32]	Use with FMRDL.
	ARM to VFP system register	FMXR{cond} <VFPsysreg>, Rd		VFPsysreg := Rd	Stalls ARM until all VFP ops complete.
	VFP system register to ARM	FMRX{cond} Rd, <VFPsysreg>		Rd := VFPsysreg	Stalls ARM until all VFP ops complete.
	FPSCR flags to CPSR	FMSTAT{cond}		CPSR flags := FPSCR flags	Equivalent to FMRX R15, FPSCR

Exceptions	
IO	Invalid operation
OF	Overflow
UF	Underflow
IX	Inexact result
DZ	Division by zero

Vector Floating Point Instruction Set
Quick Reference Card

FPSCR format								Rounding		(Stride – 1)*3			Vector length – 1				Exception trap enable bits							Cumulative exception bits				
31	30	29	28				24	23	22	21	20		18	17	16				12	11	10	9	8				4	3	2	1	0
N	Z	C	V				FZ	RMODE		STRIDE			LEN						IXE	UFE	OFE	DZE	IOE				IXC	UFC	OFC	DZC	IOC
FZ: 1 = flush to zero mode.								Rounding: 0 = round to nearest, 1 = towards $+\infty$, 2 = towards $-\infty$, 3 = towards zero.																	(Vector length * Stride) must not exceed 4 for double precision operands.				

If Fd is S0-S7 or D0-D3, operation is Scalar (regardless of vector length).	If Fd is S8-S31 or D4-D15, and Fm is S0-S7 or D0-D3, operation is Mixed (Fm scalar, others vector).
If Fd is S8-S31 or D4-D15, and Fm is S8-S31 or D4-D15, operation is Vector.	S0-S7 (or D0-D3), S8-S15 (D4-D7), S16-S23 (D8-D11), S24-S31 (D12-D15) each form a circulating bank of registers.

Condition Field		
Mnemonic	**Description (Thumb)**	**Description (VFP)**
EQ	Equal	Equal
NE	Not equal	Not equal, or unordered
CS / HS	Carry Set / Unsigned higher or same	Greater than or equal, or unordered
CC / LO	Carry Clear / Unsigned lower	Less than
MI	Negative	Less than
PL	Positive or zero	Greater than or equal, or unordered
VS	Overflow	Unordered (at least one NaN operand)
VC	No overflow	Not unordered
HI	Unsigned higher	Greater than, or unordered
LS	Unsigned lower or same	Less than or equal
GE	Signed greater than or equal	Greater than or equal
LT	Signed less than	Less than, or unordered
GT	Signed greater than	Greater than
LE	Signed less than or equal	Less than or equal, or unordered
AL	Do not use in Thumb	Always (normally omitted)

Exceptions	
IO	Invalid operation
OF	Overflow
UF	Underflow
IX	Inexact result
DZ	Division by zero

Document Number

ARM QRC 0001H

Change Log

Issue	Date	By	Change
A	June 1995	BJH	First Release
B	Sept 1996	BJH	Second Release
C	Nov 1998	BJH	Third Release
D	Oct 1999	CKS	Fourth Release
E	Oct 2000	CKS	Fifth Release
F	Sept 2001	CKS	Sixth Release
G	Jan 2003	CKS	Seventh Release
H	Oct 2003	CKS	Eighth Release

www.arm.com